第5卷 第4期 半 导 体 学 报 Vol. 5, No. 4
1984 年 7 月 CHINESE JOURNAL OF SEMICONDUCTORS July, 1984

LSI 二维布局的分析算法

周 电 唐璞山
(复旦大学电子工程系)

1983 年 6 月 25 日收到

本文提出一种适合于门阵列的布局算法，该算法采用分析方法处理二次分配的组合问题，具体使用最优相对布局及线性分配定位或快速定位方法得到一个接近最优解．实际结果表明不论初始分布如何，均可在差不多相同的时间内获得优化结果大致相同的结果．

本算法(用快速定位)的计算复杂性为 $O(n)$，计算速度很快，一般对一百至二百个单元的芯片，计算速度比通常的力交换法快一个数量级．本算法对大系统的适应性特别好．

一、引 言

本文提出一种适合于门阵列使用的布局算法，有关这方面的工作已发表了不少[1-4]．本算法的数学模型是二次分配问题[5]．由于二次分配问题的复杂性，目前还只有一些启发式的近似算法，这些算法已经在实际工作中获得了广泛的应用，但还存在一些问题：首先，解的优劣对问题本身以及初始布局的依赖程度很大；其次，算法的运算时间还是较长的，即运算的复杂性较大，那怕是构造一个初始布局，其算法复杂性也是 $O(n^2)$ 量级。面对大规模乃至超大规模集成电路的发展，人们希望有更有效的方法以适应它的要求．本文用分析的手段，建立一种算法，保证在任何初始布局下(实际上是随机产生的初始布局)都可得到较好的解．而且算法的复杂性是 $O(n)$．这两个特点使得本算法成为一种高效的、实用的算法．

二、问题的提出和算法概述

n 个元素，每两个元素之间有其相应的连线长和表征某些要求的权(也称为费用)．如何安置这 n 个元素于一个平面上，使相应的带权连线长之和最小，这就是一个二次分配问题．

设费用矩阵为：

$$W = [\rho_{ij}] \quad i = 1, 2, \cdots n, \quad j = 1, 2, \cdots n$$

距离矩阵为：

$$D = [d_{ij}] \quad i = 1, 2, \cdots n, \quad j = 1, 2, \cdots n$$

求：

$$\min L = \min \sum_{i,j} \rho_{ij} d_{P(i)P(j)}$$

这里，P 是一切可能的置换.

图 1 给出了一个 $n=9$ 的门阵列. 每个小方格看做一个元素，每个元素具有同样的大小. 并定义有相应的费用矩阵和距离矩阵. 求这些元素在给定平面上的一个放置，对这个放置有 $\min L$. 可以认为图 1 是二次分配问题的一个物理实体.

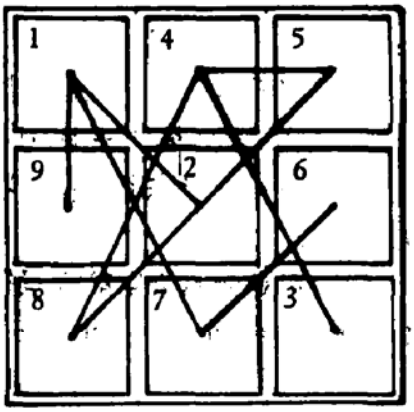

图 1

目前还没有求二次分配问题精确解的 P 算法. 对已提出的近似算法，例如 Steinberg 算法[6-7]，在每个独立集内，这个算法采用线性分配进行交换，而独立集之间的元素不能进行交换，因此不能保证解一定很好. 又如，对交换乃至三交换或更高层次的交换法[1,8]，考虑到交换是在一定层次内进行(由于算法的可行性要求，不可能进行全交换)，同样不能保证所有情况下得到好的解. 因此，这些算法是面向二次分配问题中的具体问题，而且解的优劣同初始布局关系密切. 可是人们不能在计算之前就判断什么样的问题用什么样的算法好、以及采用什么样的初始布局好等等，这在实际应用中是非常不利的. 总括说来，已有的算法共同存在的问题是：目标函数在计算过程中不一定收敛于最优解，而常常收敛于某个局部最优解，正是为此，导致了对某些问题明明距最优解还差很远，但算法却无能为力了. 本文提出一种称之谓"逼近法"的算法，对于任何问题可以使解收敛在最优解附近.

"逼近法"实质上是构造了一个迭代过程，这个过程中目标函数即总的带权长之和 L 收敛于问题的最优解附近，因而对不同的具体问题，都可获得满意的解. 算法首先进行一个放大过程，在此基础上将一个组合问题转化为一个可以用分析手段处理的问题. 接下来进行线性迭代. 考虑到原问题是有边界条件的，即规定有外形的. 我们把空间看做是与"边界"具有相同形状的由格点组成的点阵，在迭代过程中，同时对这些格点进行线性分配或快速排队，称为空间定位. 通过迭代和空间定位的交替进行得到问题的解. 必需指出核心问题是目标函数在迭代过程中表现为收敛于最优解附近.

算法的物理图象是：一个容器内装有 n 个粒子，粒子的体积大小一样，外表面理想光滑，每两个粒子间有力存在，力的大小正比于它们之间的距离和相关度，并定义有相应的体系势能. 为了使体系处于势能极小值，将容器放得足够大，粒子在容器中线性散开，然后粒子可在自由空间按其受力状况移动，同时容器逐渐收缩，当容器恢复到原大小时，可以想像，体系将达到势能最低点.

三、算法思想和实现

集成电路版图中一般采用水平和垂直两个方向的走线. 因此，可以把布局问题对应的二次分配问题表示为如下形式：

$$L=\sum_{i,j}\rho_{ij}X_{ij}+\sum_{ij}\rho_{ij}Y_{ij},\ i,j=1,2,\cdots n$$

X_{ij}、Y_{ij} 分别是元素 i 与元素 j 之间的横向距离和纵向距离，ρ_{ij} 是它们的权．我们的目标是求 $\min L$．

定义 1：当前最佳位置 r_i^*

元素 i 不受限制地可以处于版图中任一位置，同时保持其它元素不动，那么使 L 下降最大就是将元素 i 放在位置 r_i^* 上．

定义 2：当前最佳移动方向 f_i

f_i 是一个单位矢量，它定义为移动元素 i 时，其它元素不动，使得 L 有最大下降速率的方向上．

设 r_i° 是元素 i 当前位置的坐标矢量．显然，沿着 f_i 的方向将 L 对 X_i、Y_i 求偏导数，则其偏导数小于零，而且是这一点导数的极小值．

让图 1 所示的版图放大 m 倍，每个元素的位置坐标也做相应的放大，但元素本身的大小保持不变．元素放大后的位置称为空间格点．格点全体称为空间点阵．放大后带权连线长之和为 L'．图 2 示范性地给出了 $m=2$ 的情况．

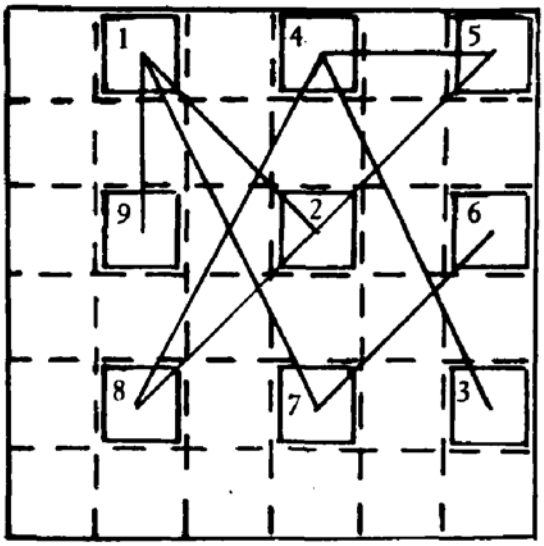

图 2

在放大后的图中，元素适当地调整相互间的位置，不要求处于格点上，只要求每个元素占据自身大小的面积，互不相交，仍然使用所给定的费用矩阵，但不受原问题版图长宽的约束，得到 $L_1=\min L'$．因为没有考虑边界条件的限制，元素是按最小带权长之和分布的，故必有 $L_1\leqslant \min L$，$\min L$ 是原问题的解．

若在上述过程中，不断插入空间定位，则定位后元素的最终位置是在格点上，获得 $L_2=\min L'$，缩小 m 倍（m 是原来放大倍数）得 $L_3=L_2/m=\min L$，如果得到 L_3 等于得到了原问题的最佳分布．

元素 i 的坐标矢量为 r_i，整个分布用 R_k 表示，脚标 k 表示第 k 次迭代后元素的分布，R 是一个 n 维空间的矢量．其关系为：

$$\left.\begin{aligned} R_k &= (r_1, r_2\cdots r_n)_k \\ \Delta R_k &= (dr_1, dr_2\cdots dr_n)_k \\ R_{k+1} &= R_k + \Delta R_k \end{aligned}\right\} \tag{1}$$

以及：

$$\left.\begin{aligned} L'_{k+1} &= L'(R_{k+1}) = L'_k + \Delta L'_k \\ \Delta L'_k &= \sum_{i=1}^{n} \Delta l_{ik} \\ \Delta l_{ik} &= (r_{ik}^* - r_{ik}^\circ)\cdot\left(\frac{\partial L'}{\partial r_i}\right)\Big|_{r_{ik}^\circ}\cdot f_{ik}\cdot dc \end{aligned}\right\} \tag{2}$$

r_{ik}° 是元素 i 的当前位置，dc 是一个很小的正数，$\left(\frac{\partial L'}{\partial r_i}\right)\Big|_{r_{ik}^{\circ}} \cdot f_{ik}$ 表示在 r_{ik}° 点沿 f_{ik} 方向求导数．由前面定义，显然有：

$$\Delta l_{ik} = -|r_{ik}^{*} - r_{ik}^{\circ}| \cdot \left|\left(\frac{\partial L'}{\partial r_i}\right)\Big|_{r_{ik}^{\circ}}\right| \cdot dc$$

亦即

$$\left.\begin{aligned} &\Delta l_{ik} \leqslant 0 \quad i = 1, 2, \cdots n \\ &\Delta L_k' = \sum_{i=1}^{n} \Delta l_{ik} \leqslant 0 \end{aligned}\right\} \tag{3}$$

由(3)式知要 $\Delta L_k' = 0$，只有当 $\Delta l_{ik} = 0$ 对全部元素成立．考虑到元素有一定大小和不允许重叠放置，所以在达到最优解之前 Δl_{ik} 总是不等于零．因此，对给定的迭代函数始终有：

$$L_{k+1}' < L_k'$$

从而保证了本算法的收敛性．

为什么在迭代过程中元素每次仅在 f_{ik} 的方向上移动一个小距离(在(2)式乘以一个小数 dc)，而不是将它直接放到 r_{ik}^{*} 所指示的位置上呢？理由是当前最佳位置并不是最终最佳位置，其它元素移动后，此位置就不再是最佳位置了．所以直接放到位置 r_{ik}^{*} 上去是没有意义的．然而，此时要回答每个元素的最终最佳位置在那里是做不到的．因此我们只要求分布 R_{k+1} 优于分布 R_k，而且每个元素在这一轮中的移动，不影响其它元素，即元素的移动互不影响，可以分别独立考虑．移动一个小量 dr_i 正是实现了这一愿望．

图 3 表示当第 k 轮迭代开始时，元素 i 处于位置 r_{ik}°，而该元素的最终最优位置是在 B，曲线表示理论上元素 i 在迭代中经历的轨迹，折线表示本算法中我们是用一系列 $dr_{ik}, dr_{i,k+1} \cdots\cdots$ 来逼近此曲线，当 dr_{ik} 足够小时，误差将是高阶小量．

图 3

元素 i 从 A 到 B 是一条曲线，反映了不断变化的分布对其迭代决策的影响．元素的分布状态不同使得元素 i 对目标函数的贡献不一．这正是解二次分配问题的难点所在．用分析的方法，取移动一个微小量后，做到了在一个微小间隔内(指一轮迭代)的线性化．这样，元素 i 对于获得 L' 的最大下降所应取的策略不受其它元素移动后的影响，产生的误差将是高阶小量，而且是不积累的．

不难看出，那些偏离最佳位置远的元素移动的步长大，偏离小的移动的步长小，甚至不移动．结果通过若干轮迭代，元素各自很快移动到了自己当前最佳位置附近．

一个重要的问题是如何考虑约束，有来自两个方面的约束．一是元素不能重叠放置，二是元素的分布边界要满足给定的边界，统称为约束条件．处理的方法是在迭代过程中插入对格点点阵的线性分配，或者用一种快速方法对元素进行空间排队，这就是空间定位．

表1 格点定位表

元素 \ 格点	1	2	·	·	j	·	·	·	·	·	n
1	r_{11}	r_{12}	·	·	r_{1j}	·	·	·	·	·	r_{1n}
2	r_{21}	r_{22}	·	·	r_{2j}	·	·	·	·	·	r_{2n}
·	·	·	·	·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	·	·	·
i	·	·	·	·	r_{ij}	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·	·	·	·
n	r_{n1}	·	·	·	r_{nj}	·	·	·	·	·	r_{nn}

表1是一张空间定位用的线性分配表. r_{ij} 表示元素 i 与格点 j 之间的曼哈顿距离.

前面已经提到,可以用一种快速空间定位的方法——排队法,这种方法是将通过迭代获得的分布 R_k 的每个元素按其位置上下左右的关系,排成空间点阵的形状. 例如原问题要求有H行、J列,就先将 R_k 中最上面的J个元素挑出来,再按这J个元素的横坐标从左到右排列,做为空间点阵的最上面一行. 然后将剩下的元素进行类似的处理,构成空间点阵的第2行、第3行、一直到做完第H行,这种方法速度快. 应该指出,在叠代过程中是有可能记录下元素的上下左右的关系,这可以避免多余的计算.

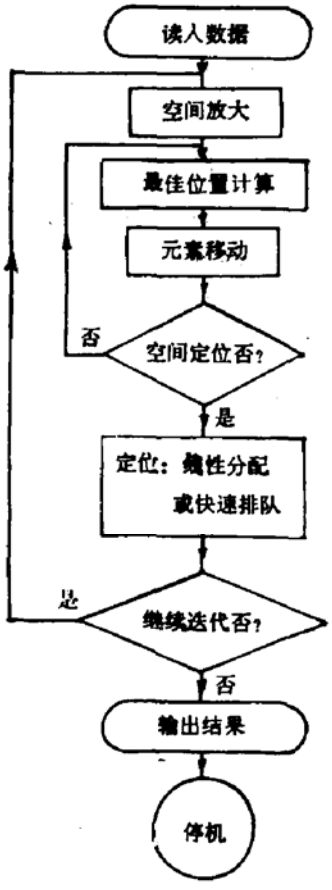

图 4

关于算法复杂性的估计. 前面提到,只要通过为数不多的若干轮迭代,各个元素就移到了自己的最佳位置附近,这种情况下,再继续迭代是意义不大的,因为在空间定位时,实际上只能做到使每个元素定位于其最佳位置附近的一个邻域内. 如果元素都到了其最佳位置附近后再继续迭代所得的好处,会在空间定位时损失,因此在这种情况下将终止迭代. 从已经介绍的步长的选取方法来看,各元素到其最佳位置附近所需的迭代次数,不是元素数目的函数(计算实例已证明了这一结论). 另外,每移动一个元素,也不需要计算它同全部其它元素的关系,只要计算它同本信号网的元素的关系即可. 根据统计资料,信号网所含有的平均元素个数,也不是总元素n的函数. 综合以上分析的几个因素,算法的复杂性为 $O(n)$.

实现本算法的程序框图如图4所示.

四、讨论和实验结果

1. 关于收敛于最优解附近的问题

如果只有迭代过程,已经证明,可以使目标函数单调下降,收敛于问题在无"外边界"

条件下的最优解．然而，当通过空间定位后，从平均意义上来说元素都被定位于其最佳位置附近的一个邻域内，因此会使目标函数有一个上升．由此，我们说目标函数在叠代过程中，在最优解附近是振荡的．在远离最优解时目标函数是单调下降的．这样使本算法对任何初始布局都可得到较好的解．图 5 和图 6 分别表示交换法和本算法的目标函数对于

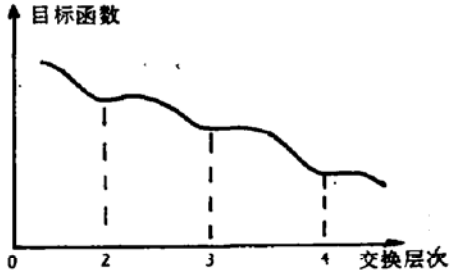

图 5

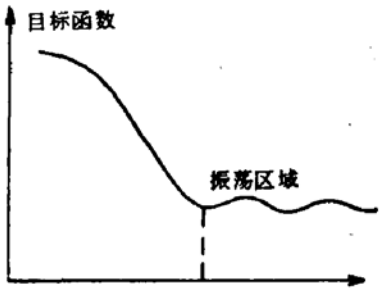

图 6

迭代层次或迭代过程的变化曲线，以形象地说明两种算法在收敛性质上的差别．

2. 算法的适应性问题

设一元素所占的面积同芯片的总面积之比为 β．当 β 较大时，算法主要工作在振荡区域．这种情况下，振荡将使得目标函数在较差的解与较好的解之间取值不定．很难保证解一定较好，这就是本算法对小问题的不适应性．当 β 较小时，进入振荡区域后，其结果在一个好的解附近摆动，不会使结果远离该解．显然，问题愈大，β 愈小，计算结果愈好，这就是本算法对大问题的适应性．

3. 关于当前最佳位置 r_i^* 的问题

力向量布局法是把连线看成弹簧，把相应的权看做弹性系数，其目标函数使各元素所受力之和最小．这种方法把元素受的合力指示的位置做为交换时的目标位置．应该指出这是欠妥当的．图 7 和图 8 给出了一个简单的例子：它有一个可动元素（元素 1）和两个

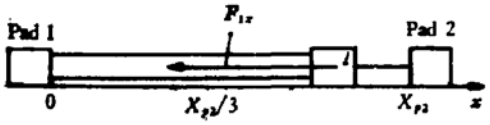

图 7

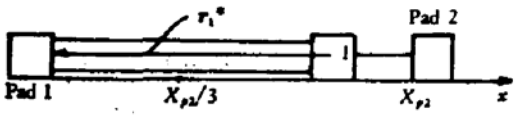

图 8

固定引线脚，设 $\rho \equiv 1$，在一维情况下求解．按力向量布局法，元素 1 受的合力指出的位置为 $x_1 = x_{p2}/3$ 处．迭代的最终结果也是使元素 1 处于 $x_1 = x_{p2}/3$ 处．显而易见，$x_1 = 0$ 是本问题的最优解． 当元素 1 初始落在区域 0 到 $x_{p2}/3$ 内时，将发生元素按力指向移动，然而解却愈来愈差，这显然是不能接受的．而按前面定义的当前最佳位置 r_i^* 移动，因为每一轮迭代中，r_i^* 都指向 $x_1 = 0$ 的位置，因此迭代的结果是 $x_1 = 0$．所以我们的算法中不用力向量指的位置作为目标函数而是用当前最佳位置 r_i^*．

4. 与布线程序互联问题

在集成电路 CAD 系统中，布局程序和布线程序总是依次运行．如果布线有问题（即

某些线段布不下)，可以回到布局程序处理. 这就要求对于大问题有一个快速的布局方法，否则很难多次处理. 本算法对于 LSI 和 VLSI 布局布线系统特别适合.

5. 运算中的一些具体问题

实现本算法时，可以考虑采用实型变量而不进行空间放大，另外允许元素交叠，利用空间定位将其分离开. 关于引线脚，可以作为不能移动的元素来处理. 计算 $\frac{\partial L}{\partial x_{ik}}$、$\frac{\partial L}{\partial y_{ik}}$ 用下列公式:

$$\frac{\partial L}{\partial x_{ik}} = \left[\left(\sum_{j=1}^{n}\rho_{ij}\right)_{x_j>x_i} - \left(\sum_{j=1}^{n}\rho_{ij}\right)_{x_j<x_i}\right]_k - [\delta\rho_{ix}]_k$$

$$\frac{\partial L}{\partial y_{ik}} = \left[\left(\sum_{j=1}^{n}\rho_{ij}\right)_{y_j>y_i} - \left(\sum_{j=1}^{n}\rho_{ij}\right)_{y_j<y_i}\right]_k - [\delta\rho_{iy}]_k$$

计算 $r_{ik}^* = x_{ik}^* + y_{ik}^*$ 可用下列表达式:

用 $\min[\delta\rho_{ix}]_k$ 确定 x_{ik}^*

用 $\min[\delta\rho_{iy}]_k$ 确定 y_{ik}^*

本算法已经用 FORTRAN-IV 编写成程序，并已在 PDP-11/34 机上实现. 表 2 给出

表 2 实例计算结果

芯片编号	元素数	$\bar{L}_0$	$\bar{L}'$	T (秒)
1#	64	1150	560	150
2#	141	7100	3540	400

两个例子的计算结果. 芯片 1 是一个有 64 个元素的随机逻辑电路; 芯片 2 是一个有 141 个元素的随机逻辑电路，并有 25 个引线脚. 表中 $\bar{L}_0$ 是由随机器产生的 60 个初始布局的平均长度，L_0 的偏离范围一般不大于 4%，$\bar{L}'$ 是对每个初始布局进行迭代计算及定位后的平均长度，L' 的变化范围小于 3.5%. 同一些已发表的计算方法比较，优化程度大致相同.

本算法不需要构造一个初始布局，这对提高速度非常有利. 因为构造一个初始布局所用时间已经大大超出本算法所用的时间. 此外，当从布线程序返回时，由随机布局出发，可以产生一个连线总长度基本相同的新布局，有利于再次布线.

田曾举、汪庆、陈乃群、陈伟等同志对本文提出一些有益的看法;张钦海同志为本算法编写了部分程序;侯加敏、钱佩华同志在调试程序时给以支持，在此特致谢意.

参 考 文 献

[1] S. Goto, Proc. of 16th Design Automation Conference, pp 11—12 (1979).

[2] H. Shiraishi and F. Hirose, Proc. of 17th Design Automation Conference, pp 458—464 (1980).

[3] K. H. Kho Khami *et al.*, Proc. of 18th Design Automation Conference, pp 426—434 (1981).

[4] N. R. Quinn, JR. M. and A. Breuer, *IEEE Trans. on Circuit and System*, CAS-26, pp 377—387 (1979).

[5] 布鲁尔(美)著，数字计算机设计自动化的理论和方法，科学出版社，北京(1978).

[6] L. Steinberg, *SIAM Rev.* 3, 36 (1961).
[7] H. W. Carter, M. A. Breuer, and Z. A. Sged, Proc. of 16th Design Automation Conference, pp. 26—31 (1979).
[8] M. Hanan, P. K. Wolf and B. J. Angali, Proc. of 13th Design Automation Conference, pp. 214—224 (1976).

An Analytic Algorithm for Two Dimensional Placement of LSI

Zhou Dian and Tang Pushan
(Department of Electronics Engineering, Fudan University)

Abstract

This paper presents an algorithm dealing with the placement problem of Gate-Array chip. The algorithm applies an analytic method to the problem of quadratic assignment which is usually treated as a combinatorial problem. The optimum relative placement method, linear-assingment or fast space determination are used in the algorithm to find a near-best solution. The calculated results show that approximately the same minimized solution which is independent of initial placement can be obtained by the algorithm at about the same time interval.

The complexity of the algorithm (using fast space determination) is proportional to the order of n., i.e. $O(n)$. It is generally very fast. For problems containing about 100 to 200 cells, the calculating speed is about ten times faster than that of Forced Directed Placement. And the algorithm is quite suitable to large scale automatic design system.